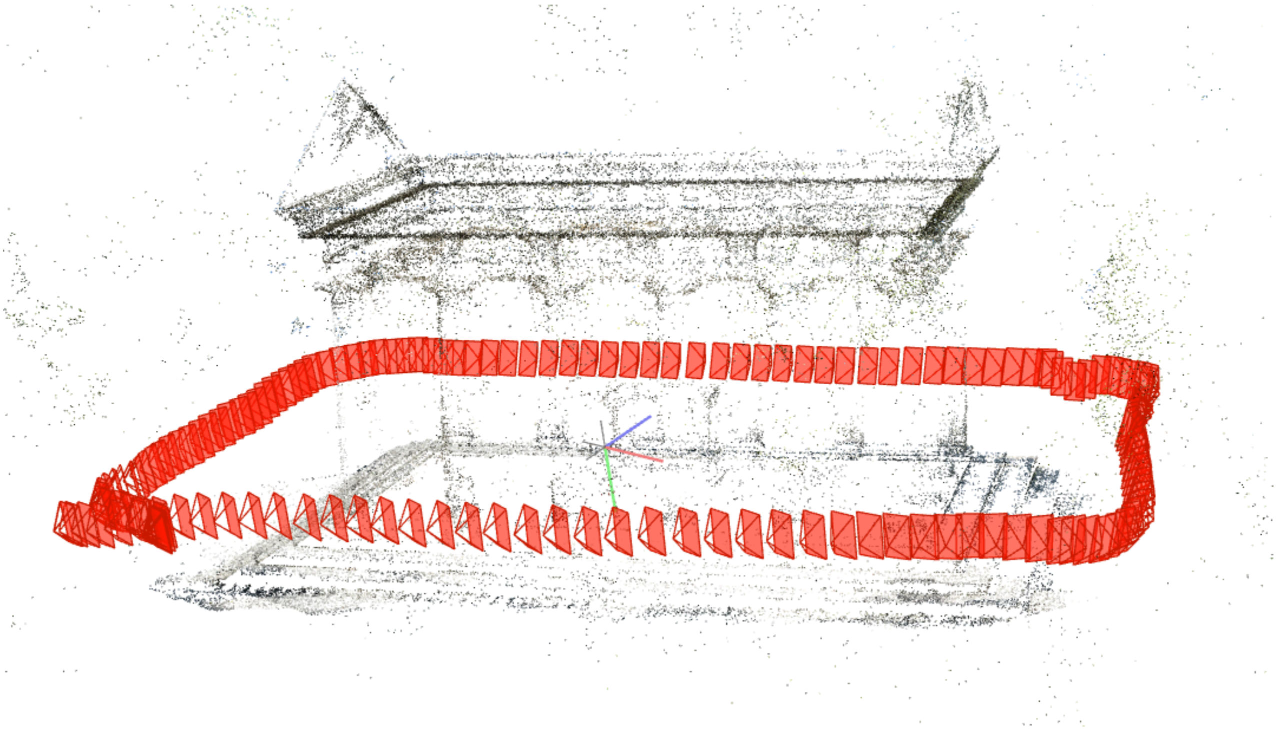
3DCV HW2 Report

Problem 1

Q1-1

先拍攝一段台大校園的影片，這邊是使用繞傅園一圈的影片，之後使用VLC media player內建的功能每10個 frame 擷取一張照片，得到不同角度共164張相片。

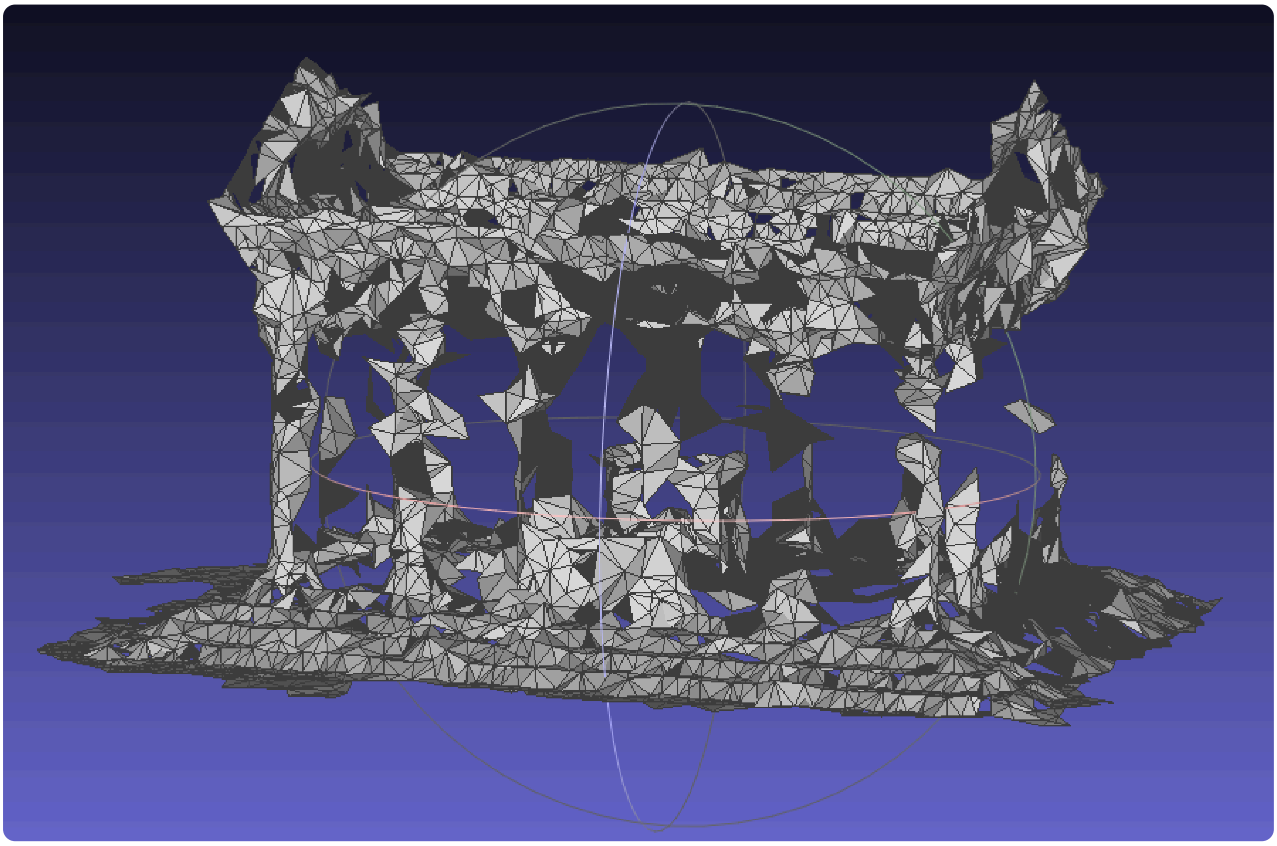
直接使用 COLMAP 的 auto reconstruction 功能，選擇這一組 data 作為影像來源，就能夠得到重建後的點雲模型，這裡只有實作 sparse reconstruction 。



在這個模型中可以發現，屋頂的點較少，反應出影片沒有拍攝到屋頂的角度。

Q1-2

將 COLMAP 得到的點雲匯入 3D processing 的軟體當中（這邊使用 meshlab），並且手動將周遭非建築的點刪除之後，我先嘗試使用了 point cloud simplification 的功能，降低點的密度後（嘗試調整降低的程度），再使用 ball pivoting 的方式做 surface reconstruction，得到以下的結果。



P1 Demo Video Link: https://youtu.be/7Lq5HW2hn_s (https://youtu.be/7Lq5HW2hn_s)

Problem 2

Environment

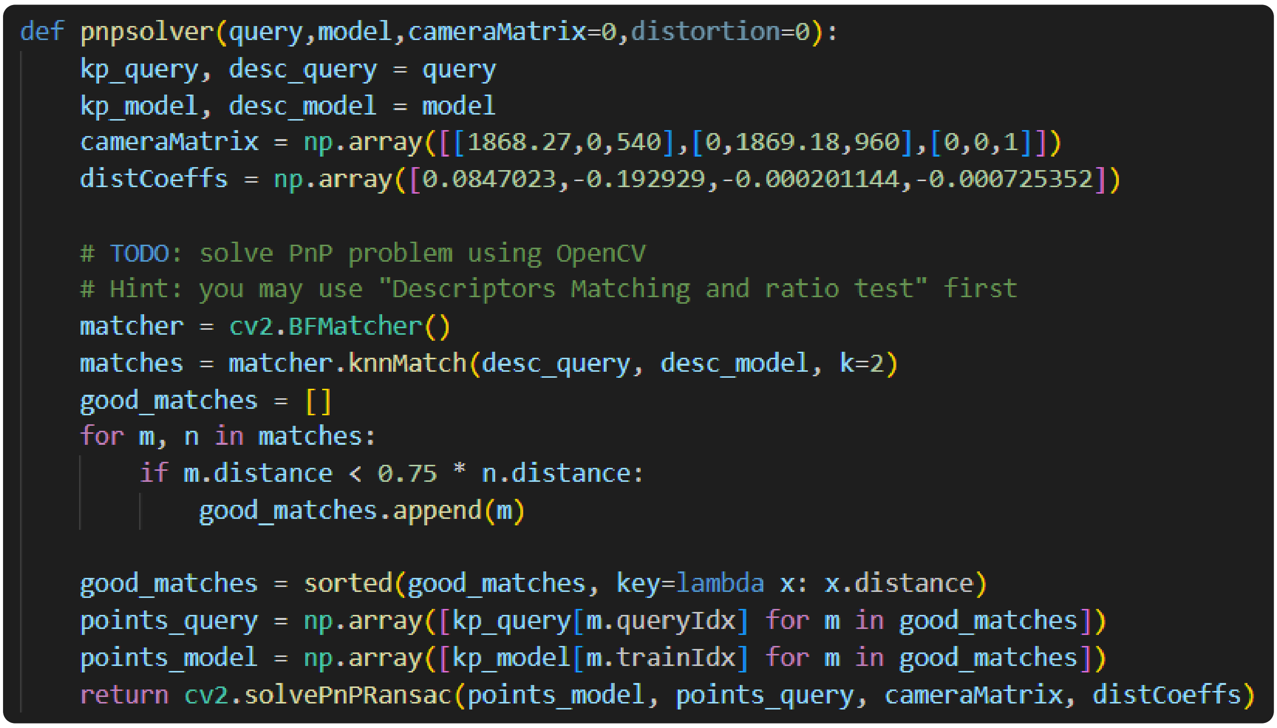
- Python == 3.11.5
- numpy == 2.2.6
- open3d == 0.19.0
- opencv-python == 4.12.0.88
- pandas == 2.3.3
- scipy == 1.16.2
- tqdm == 4.67.1

Code Execution

Please check `data` , `cube_transform_mat.npy` , `cube_vertices.npy` are in the same directory and directly run `2d3dmathcing.py` to show open3D result, and the AR cube video will be output as `ar_cube_video.mp4` in the same directory after the program finished.

Q2-1

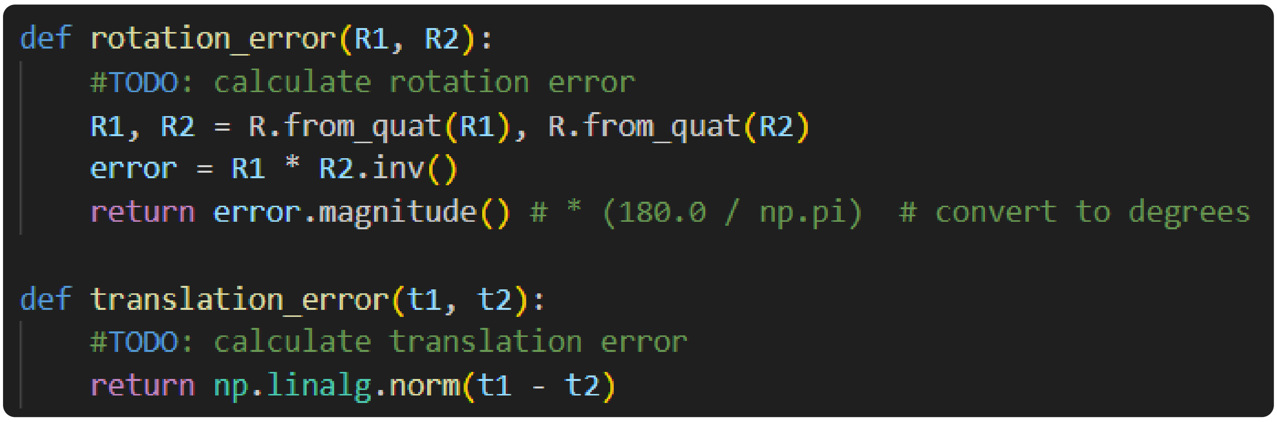
Step 1: pnpsolver



```
def pnpsolver(query,model,cameraMatrix=0,distortion=0):
    kp_query, desc_query = query
    kp_model, desc_model = model
    cameraMatrix = np.array([[1868.27,0,540],[0,1869.18,960],[0,0,1]])
    distCoeffs = np.array([0.0847023,-0.192929,-0.000201144,-0.000725352])

    # TODO: solve PnP problem using OpenCV
    # Hint: you may use "Descriptors Matching and ratio test" first
    matcher = cv2.BFMatcher()
    matches = matcher.knnMatch(desc_query, desc_model, k=2)
    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)

    good_matches = sorted(good_matches, key=lambda x: x.distance)
    points_query = np.array([kp_query[m.queryIdx] for m in good_matches])
    points_model = np.array([kp_model[m.trainIdx] for m in good_matches])
    return cv2.solvePnPRansac(points_model, points_query, cameraMatrix, distCoeffs)
```

這裡使用跟 HW1 相同的方式尋找 good_matches，並且直接利用 openCV 的 function `solvePnPRansac` ，就能夠得到每個影像在 world coordinate system 下的 camera pose。

Step 2: compute the median pose error



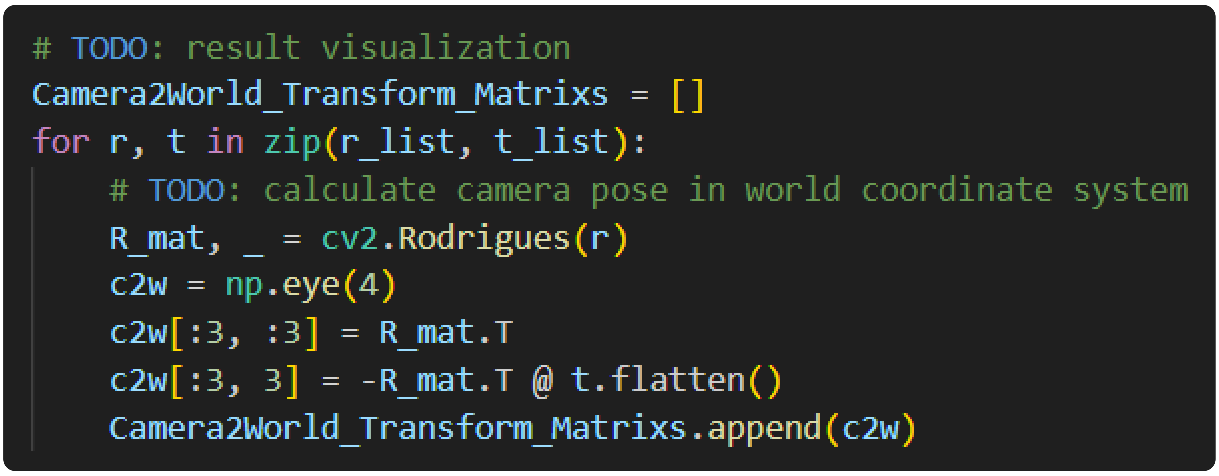
```python
def rotation_error(R1, R2):
    #TODO: calculate rotation error
    R1, R2 = R.from_quat(R1), R.from_quat(R2)
    error = R1 * R2.inv()
    return error.magnitude() # * (180.0 / np.pi)  # convert to degrees

def translation_error(t1, t2):
    #TODO: calculate translation error
    return np.linalg.norm(t1 - t2)
```

在 rotation error 的部分，利用 axis angle representation 計算兩者的 relative rotation `R1 * R2.inv()` 。

Translation error 則可以直接用 Euclidean Distance 表示。

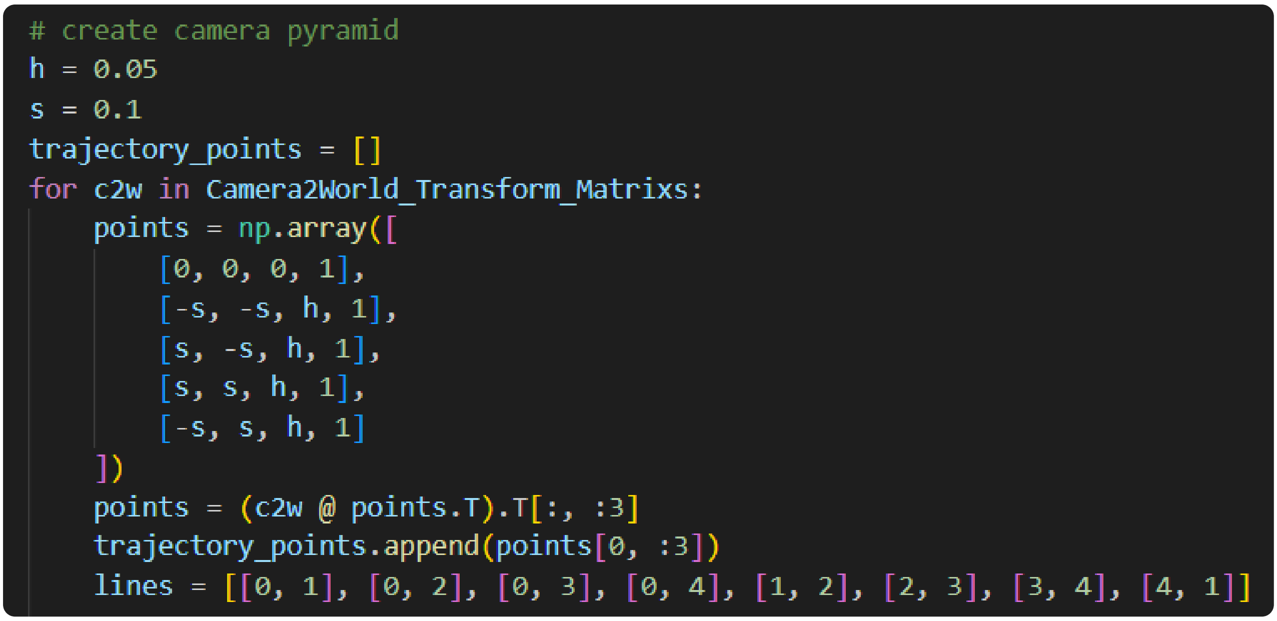
```
rotation differences:  4.2022682564173055e-05
translation differences:  0.00012142315972508917
```

Step 3: plot the trajectory and camera poses



```python
# TODO: result visualization
Camera2World_Transform_Matrixs = []
for r, t in zip(r_list, t_list):
    # TODO: calculate camera pose in world coordinate system
    R_mat, _ = cv2.Rodrigues(r)
    c2w = np.eye(4)
    c2w[:3, :3] = R_mat.T
    c2w[:3, 3] = -R_mat.T @ t.flatten()
    Camera2World_Transform_Matrixs.append(c2w)
```

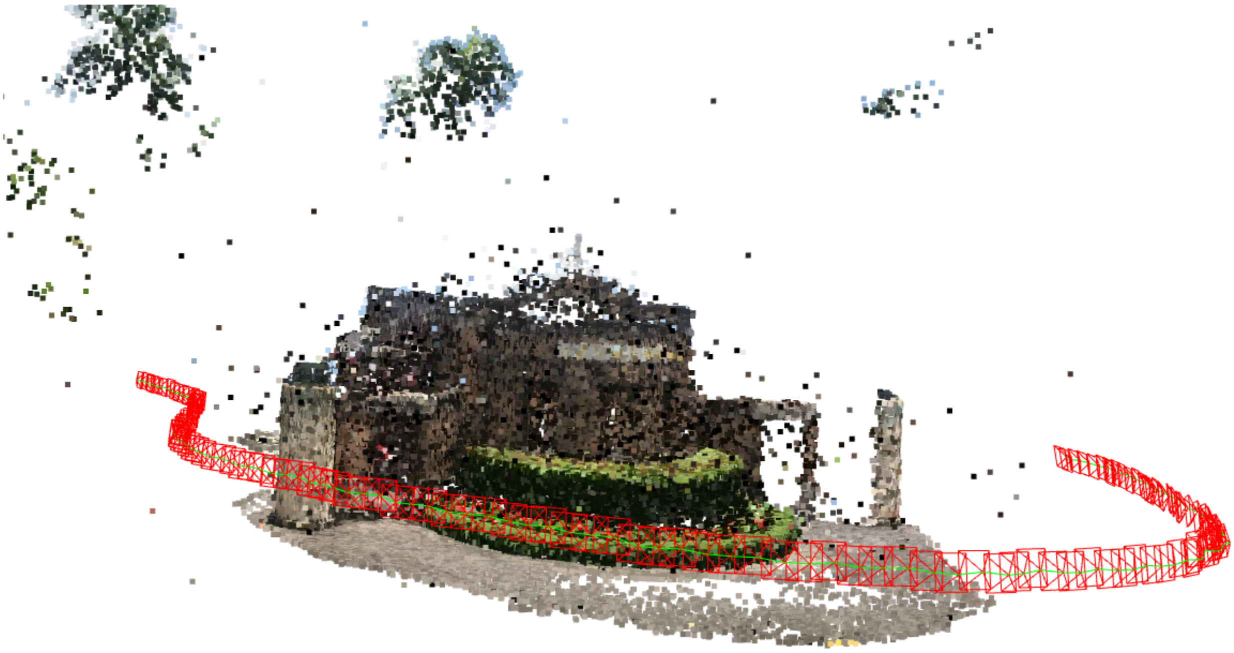
把各個 camera pose 的 w2c matrix 轉成 c2w matrix：[R^T | -R^T * t]



```python
# create camera pyramid
h = 0.05
s = 0.1
trajectory_points = []
for c2w in Camera2World_Transform_Matrixs:
    points = np.array([
        [0, 0, 0, 1],
        [-s, -s, h, 1],
        [s, -s, h, 1],
        [s, s, h, 1],
        [-s, s, h, 1]
    ])
    points = (c2w @ points.T).T[:, :3]
    trajectory_points.append(points[0, :3])
    lines = [[0, 1], [0, 2], [0, 3], [0, 4], [1, 2], [2, 3], [3, 4], [4, 1]]
```

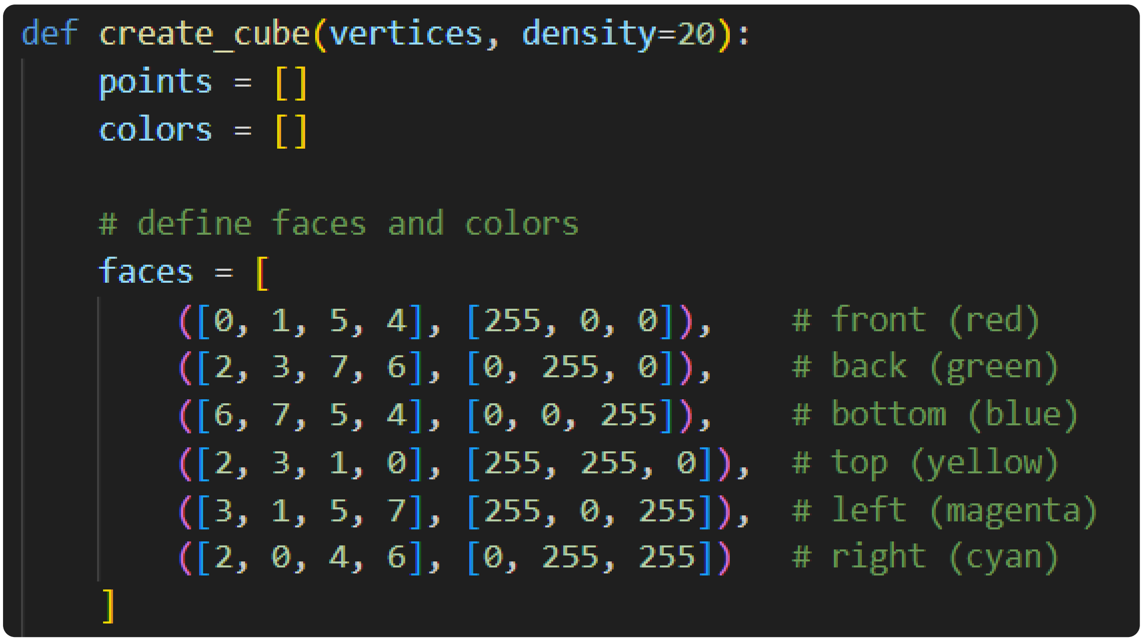
先將點雲讀進 open3D，之後在 CCS 上以 camera 為原點把 camera pyramid 的點找好之後，用上面得到的 c2w matrix 將所需的點投影到 WCS 上，連線後得到 camera pyramid。並且將各個 camera pose 連線得到 trajectory line，最後在模型上一起顯示。

（注意 IMAGE_ID 跟相機軌跡的順序可能不同，這裡有利用檔案名稱的數字重新排序）



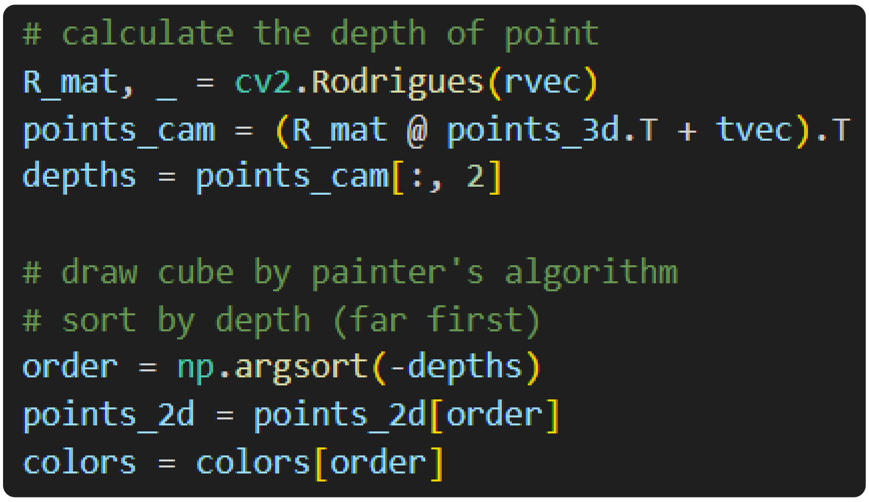
Q2-2

先利用助教提供的程式能夠得到 cube transform matrix 及八個頂點的 vetrices。



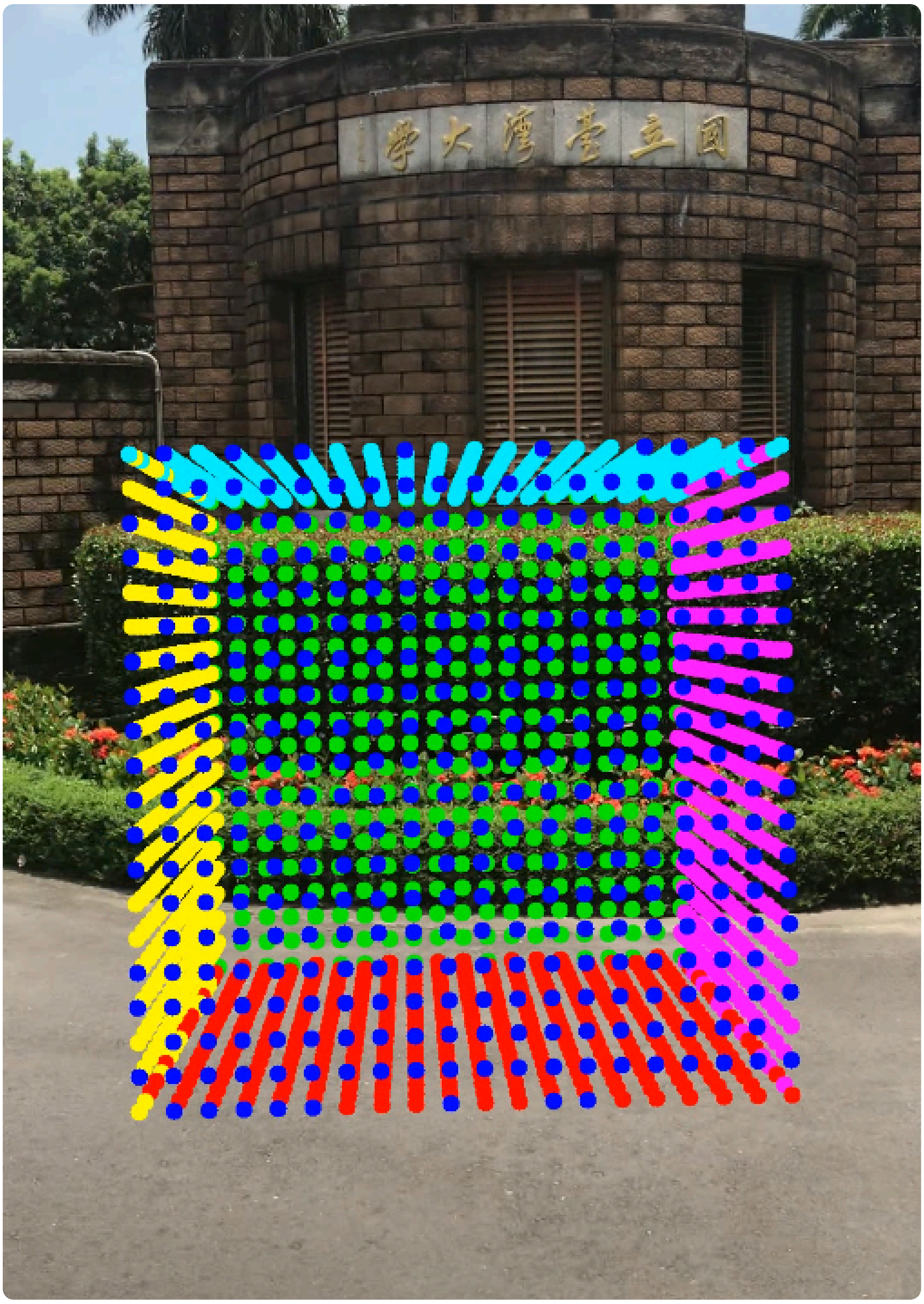
```
def create_cube(vertices, density=20):
    points = []
    colors = []

    # define faces and colors
    faces = [
        ([0, 1, 5, 4], [255, 0, 0]),     # front (red)
        ([2, 3, 7, 6], [0, 255, 0]),     # back (green)
        ([6, 7, 5, 4], [0, 0, 255]),     # bottom (blue)
        ([2, 3, 1, 0], [255, 255, 0]),   # top (yellow)
        ([3, 1, 5, 7], [255, 0, 255]),   # left (magenta)
        ([2, 0, 4, 6], [0, 255, 255])    # right (cyan)
    ]
```

先指定正方體的六個面及對應個顏色（注意頂點的順序），之後對於各個面用 bilinear interpolation 的方式將點填滿，將得到的點乘上 cube transform matrix 得到各點的座標，再對於每個影像利用 openCV funtion `projectPoints` 得到投影到各個影像上的座標。



```
# calculate the depth of point
R_mat, _ = cv2.Rodrigues(rvec)
points_cam = (R_mat @ points_3d.T + tvec).T
depths = points_cam[:, 2]

# draw cube by painter's algorithm
# sort by depth (far first)
order = np.argsort(-depths)
points_2d = points_2d[order]
colors = colors[order]
```

計算各個點對於 camera 的深度，在各個影像上實作 painter's algorithm (較遠的點先畫)，並且只畫包含在影像內的點，最後將影像合起來得到 AR cube video。



（影片截圖，完整影片連結在下方）

P2 Demo Video: https://youtu.be/u_0Q8VOcyl8 (https://youtu.be/u_0Q8VOcyl8) (Note: Due to video length consideration, demo video only used 31 pictures for demonstration. The full AR cube video is below.)

Full AR cube Video: https://youtube.com/shorts/vr1h5UK_MzE (https://youtube.com/shorts/vr1h5UK_MzE)

LLM use: Used Github Copilot for debugging (including checking matrix computation correctness,) open3D/openCV usage (including adding items on existing videos or images.)